BPTT and Complex Architectures

AW

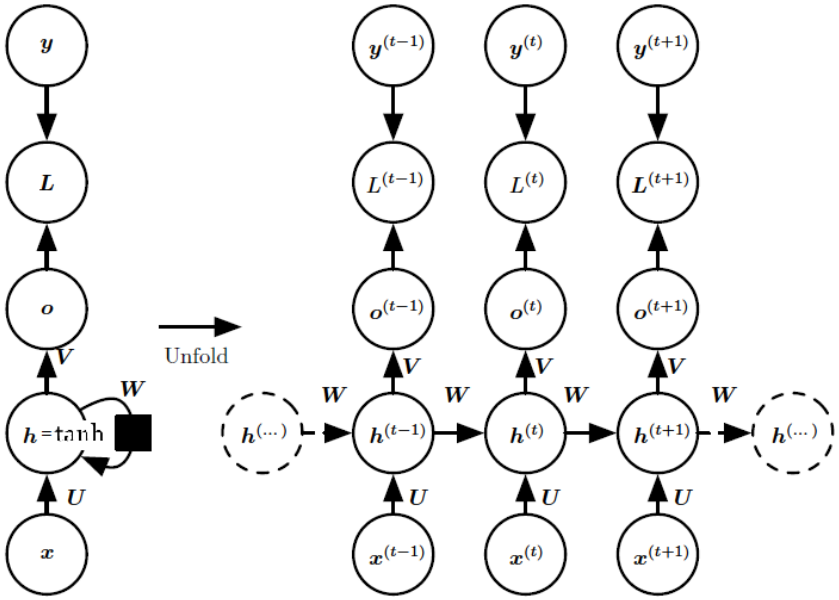
Lecture Overview

1. BPTT
2. Complex Networks
3. Exploding and Vanishing Gradients
4. Long Term Dependencies

Back Propagation in Time by Example

Example RNN:

$$\vec{a}^{(t)} = \vec{b} + W\vec{h}^{(t-1)} + U\vec{x}^{(t)}$$
$$\vec{h}^{(t)} = \tanh(\vec{a}^{(t)})$$
$$\vec{o}^{(t)} = \vec{c} + V\vec{h}^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}(\vec{o}^{(t)})$$
$$L(\mathrm{x}, \mathrm{y}) = -\sum_{t=1}^{\tau}\sum_{i=1}^{k} y^t \log \hat{y}_i^{(t)}$$



Where $y_i^{(t)} \in \{0,1\}$ and for each t

- There exists only one i such that $y_i^{(t)} = 1$
- for all $j \neq i$ we have $y_j^{(t)} = 0$ ($\vec{y}^{(t)}$ is one-hot vector for all $t \in \{0, \dots, \tau\}$)

Goal and Known Facts

Desiderata:

- The nodes of our computational graph include parameters U , V , W , $\vec{b}$ and $\vec{c}$ as well as the sequence of nodes indexed by t for $\vec{x}^{(t)}$, $\vec{h}^{(t)}$, $\vec{o}^{(t)}$ and $L^{(t)}$, so for each node N we need to compute gradient with respect to all these parameters which I denote $\nabla_N L$

Known quantities:

- We have already computed forward propagation pass so for every time $t \in \{1, \dots, \tau\}$ we have computed values of $\hat{y}_i^{(t)}$ and $\vec{o}^{(t)}$
- Gradient of cross entropy for softmax is $-\frac{y_i}{\hat{y}_i}$ where is a ground truth probability of class i (1 if observed, 0 otherwise)
- Gradient for tanh is $1 - \hat{\mathrm{y}}^2$ (since $\tanh'(z) = 1 - \tanh^2(z)$)

Computation of Gradient For Output

- For every node N in any time slice computation of necessary gradients start with $\frac{\partial L}{\partial L^{(t)}} = \frac{\partial\left(\sum_{t=1}^{\tau} L^{(t)}\right)}{\partial L^{(t)}} = 1$
- Chain rule gives:

$$\left(\nabla_{\vec{o}^{(t)}} L\right)_i = \frac{\partial L}{\partial L^{(t)}} \cdot \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \frac{\partial\left(-\sum_{i=1}^{k} y_i^t \log \hat{y}_i^{(t)}\right)}{\partial o_i^{(t)}} = -y_i^{(t)} / \hat{y}_i^{(t)}$$

- So far no backpropagation in time necessary, but for gradients of hidden-to-hidden connections we need to start working backwards from the last hidden node as suggested by unrolled graph:

$$\nabla_{\vec{h}^{\tau}} L = \frac{\partial L}{\partial L^{(\tau)}} \cdot \frac{\partial L^{(\tau)}}{\partial \vec{o}^{(\tau)}} \cdot \frac{\partial \vec{o}^{(\tau)}}{\partial \vec{h}^{(\tau)}} \text{ where } \frac{\partial \vec{o}^{(\tau)}}{\partial \vec{h}^{(\tau)}} = V \text{ so}$$

$$\nabla_{\vec{h}^{(\tau)}} L = V^T \cdot \frac{\partial L}{\partial L^{(\tau)}} \cdot \frac{\partial L^{(\tau)}}{\partial \vec{o}^{(\tau)}} = V^T \nabla_{\vec{o}^{(\tau)}} L$$

We can now recurse from $t-1$ to 0 since

$$\nabla_{\vec{h}^{(t)}} L = \frac{\partial L}{\partial \vec{h}^{(t+1)}} \cdot \frac{\partial \vec{h}^{(t+1)}}{\partial h^{(t)}} + \frac{\partial L}{\partial L^{(t)}} \cdot \frac{\partial L^{(t)}}{\partial \vec{o}^{(t)}} \cdot \frac{\partial \vec{o}^{(t)}}{\partial \vec{h}^{(t)}}$$

Computation of Gradient for Hidden Nodes

$$\nabla_{\vec{h}^{(\tau)}} L = V^T \cdot \frac{\partial L}{\partial L^{(\tau)}} \cdot \frac{\partial L^{(\tau)}}{\partial \vec{o}^{(\tau)}} = V^T \nabla_{\vec{o}^{(\tau)}} L$$

We can now recurse from $t-1$ to 0 since

$$\nabla_{\vec{h}^{(t)}} L = \frac{\partial L}{\partial \vec{h}^{(t+1)}} \cdot \frac{\partial \vec{h}^{(t+1)}}{\partial h^{(t)}} + \frac{\partial L}{\partial L^{(t)}} \cdot \frac{\partial L^{(t)}}{\partial \vec{o}^{(t)}} \cdot \frac{\partial \vec{o}^{(t)}}{\partial \vec{h}^{(t)}}$$

$$= \left(\frac{\partial \vec{h}^{(t+1)}}{\partial h^{(t)}}\right)^T \cdot \nabla_{\vec{h}^{(t+1)}} L + V^T \nabla_{\vec{o}^{(t)}} L$$

We have $\frac{\partial \vec{h}^{(t+1)}}{\partial h^{(t)}} = \text{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t+1)}}\right)^2\right) W$ where $\hat{y}_{\vec{h}^{(t+1)}}$ is the outputs of hidden $(t+1)$ nodes, and $\text{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t+1)}}\right)^2\right)$ is a matrix with values in parenthesis computed on forward pass on the diagonal (Jacobian of tanh) and 0's everywhere else. So

$$\nabla_{\vec{h}^{(t)}} L = W^T \cdot \nabla_{\vec{h}^{(t+1)}} L \cdot \text{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t+1)}}\right)^2\right) + V^T \nabla_{\vec{o}^{(t)}} L$$

Computation of Gradient for Parameters

- Once we have hidden nodes we have a path to every parameter.

Problem: parameters are not matrices but 3D tensors with one dimension being time, e.g. we do not have W but we do have $W^{(t)}$. But we want them all to be shared= equal, e.g. $W^{(t_1)}$ must be equal to $W^{(t_2)}$ for all t_1, t_2. This isn't going to happen with back propagation.

Solution: We want the gradient of shared parameters to have contribution of all edges that lead to this parameter shared parameters so the total gradient is just the sum of gradients for all times, e.g. $\nabla_W L = \sum_{t=1}^{\tau} \nabla_{w^{(t)}} L$

Gradient for Parameters (cont.)

- We have $\nabla_W L = \sum_{t=1}^{\tau} \nabla_{w^{(t)}} L$ and also

$$\nabla_{w^{(t)}} L = \sum_i \frac{\partial L}{\partial h_i^{(t)}} \nabla_{W^{(t)}} h_i^{(t)}$$

$$= \mathrm{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t)}}\right)^2\right) \cdot \left(\nabla_{\vec{h}^{(t)}} L\right) \cdot \left(\vec{h}^{(t-1)}\right)^T$$

then

$$\nabla_W L = \sum_{t=1}^{\tau} \nabla_{W^{(t)}} L$$

$$= \sum_{t=1}^{\tau} \mathrm{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t)}}\right)^2\right) \cdot \left(\nabla_{\vec{h}^{(t)}} L\right) \cdot \left(\vec{h}^{(t-1)}\right)^T$$

Gradient for Parameters (cont.)

Similarly

- $\nabla_{\vec{c}} L = \sum_{t=1}^{\tau} \nabla_{\vec{c}^{(t)}} L = \sum_{t=1}^{\tau} \left(\frac{\partial \vec{o}^{(t)}}{\partial \vec{c}} \right)^T \nabla_{\vec{o}^{(t)}} L = \sum_{t=1}^{\tau} \nabla_{\vec{o}^{(t)}} L$
- $\nabla_{\vec{b}^{(t)}} L = \sum_{t=1}^{\tau} \nabla_{\vec{b}^{(t)}} L = \sum_{t=1}^{\tau} \sum_i \frac{\partial L}{\partial h_i^{(t)}} \nabla_{\vec{b}^{(t)}} h_i^{(t)}$

  $= \sum_{t=1}^{\tau} \mathrm{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t)}}\right)^2\right) \cdot \left(\nabla_{\vec{h}^{(t)}} L\right)$
- $\nabla_{V^{(t)}} L = \sum_{t=1}^{\tau} \nabla_{V^{(t)}} L = \sum_{t=1}^{\tau} \sum_i \frac{\partial L}{\partial o_i^{(t)}} \nabla_{V^{(t)}} o_i^{(t)}$

  $= \sum_{t=1}^{\tau} \nabla_{V^{(t)}} \vec{o}^{(t)} \cdot \left(\hat{y}_{\vec{h}^{(t)}}\right)^T$
- $\nabla_{U^{(t)}} L = \sum_{t=1}^{\tau} \nabla_{U^{(t)}} L = \sum_{t=1}^{\tau} \sum_i \frac{\partial L}{\partial h_i^{(t)}} \nabla_{U^{(t)}} h_i^{(t)}$

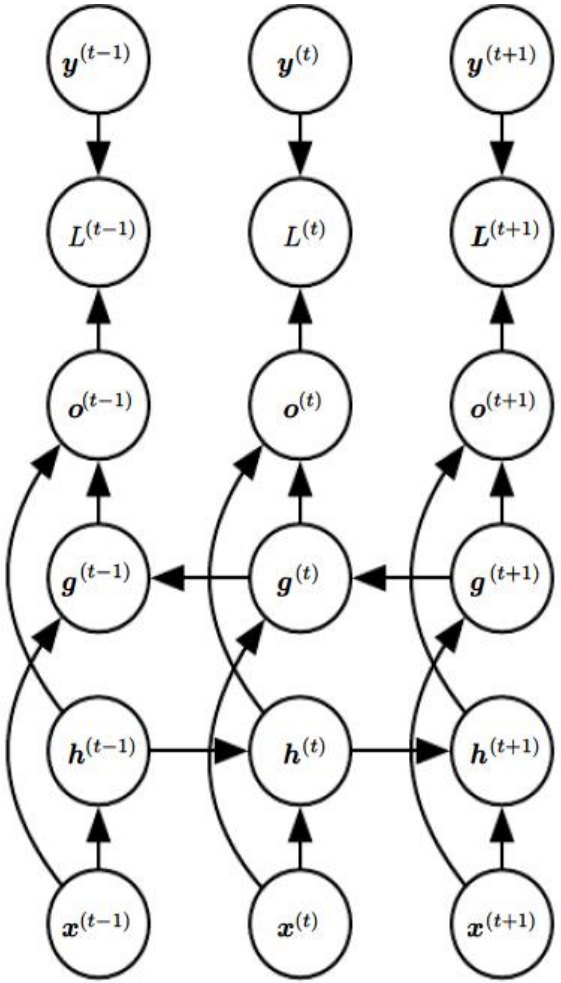
  $= \sum_{t=1}^{\tau} \mathrm{diag}\left(1 - \left(\hat{y}_{\vec{h}^{(t)}}\right)^2\right) \cdot \left(\nabla_{\vec{h}^{(t)}} L\right) \cdot \left(\vec{x}^{(t)}\right)^T$

Lecture Overview

1. BPTT
2. Complex Networks
3. Exploding and Vanishing Gradients
4. Long Term Dependencies

Bidirectional Networks

- Networks so far: output $\hat{y}^{(t)}$ may depend on all previous/current inputs $\vec{x}^{(\tau)}$, for $\tau \leq t$ for and even on true outputs $y^{(\tau)}$, but not on the whole sequence.
- Yet in many cases we need sequence to sequence tasks where output in each step depends on the whole sequence input e.g. handwriting recognition
- The answer is bidirectional RNNs in which sequence is processed from both ends: hidden units $\vec{h}^{(t)}$ propagate information forward while hidden units $\vec{g}^{(t)}$ propagate information back in time
- At time t output units $\vec{o}^{(t)}$ can use 'summary of the past' from $\vec{h}^{(t)}$ and 'summary of the future' from $\vec{g}^{(t)}$



Deep Recurrent Networks

RNNs perform 3 parameterized transformations:

1. from the input to the hidden state
2. from the previous hidden state to the next hidden state
3. from the hidden state to the output.

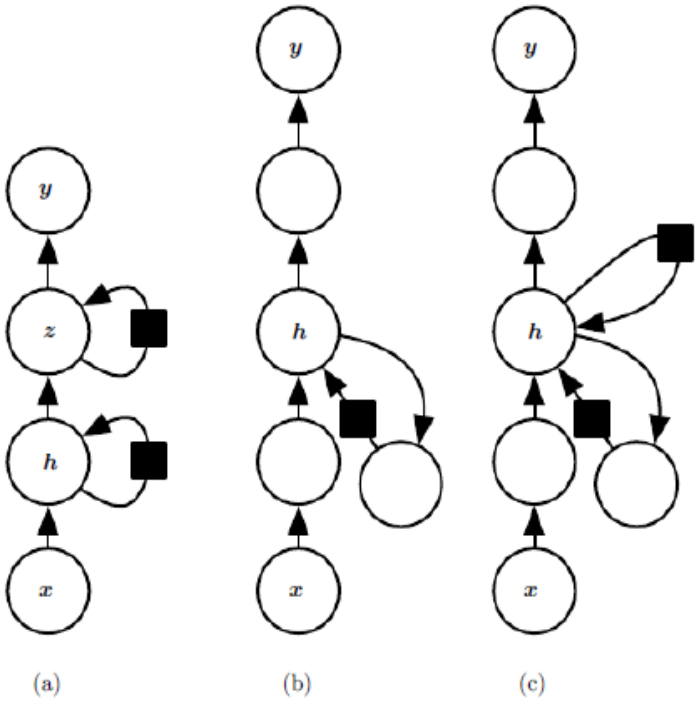
- Previous networks are shallow: when unrolled from 1 hidden layer with affine transformation followed by non-linear activation
- Would deeper unrolled networks help?
  - Introducing deeper levels is at the expense of computation of backpropagation, so will it help?
- How to add layers?

Deep Recurrent Networks: 3 Ways

a) Hierarchical addition: the hidden recurrent state broken down into groups organized hierarchically; lower layers transform the raw input into a representation that is more appropriate, and it is used at the higher levels of the hidden state

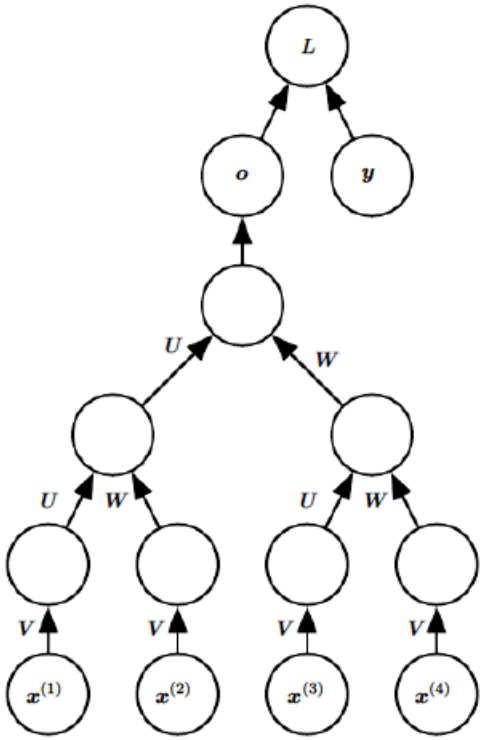
b) Separate FFN (possibly deep) for each of the three transformations. To be useful needs enough capacity in each of these three steps, but doing so by adding depth to paths may hurt backpropagation

c) Corrects problems of MLP by adding skip connections shortening the paths for approximated computations



Recursive Neural Networks

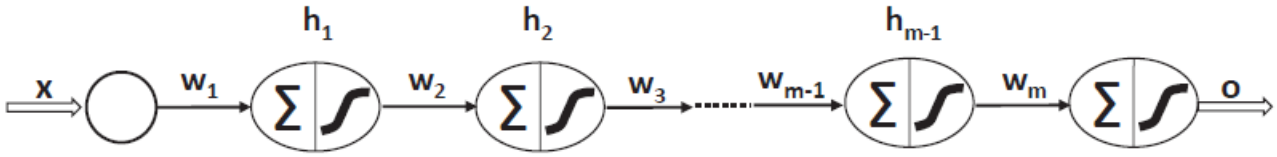
- RNNs had folded graphs that were single chain with a time-delay loop that results in a path unfolded graph
- Recursive NNs have unrolled graphs that are trees
- advantage of recursive NN over RNN is that for a sequence of the same length τ, the depth (measured as the number of compositions of nonlinear operations) can is reduced from τ to $O(\log \tau)$, which might help deal with long-term dependencies.
- What should be tree structures?
  - Data independent e.g. binary balanced tree
  - Application/data dependent, e.g. parse tree in NLP



Lecture Overview

1. BPTT
2. Complex Networks
3. Exploding and Vanishing Gradients
4. Long Term Dependencies

Intuition Behind Gradient Problems



- Neural network with one node per layer.
- Forward propagation multiplicatively depends on each weight and activation function evaluation.
- Backpropagated partial derivative get multiplied by weights and activation function derivatives.
- Unless the values are exactly one, the partial derivatives will either continuously increase (explode) or decrease (vanish).
- Hard to initialize weights exactly right.

More on Origins of Gradients' Problems

- Recurrent networks involve the composition of the same function multiple times, once per time step:

$$\vec{h}^{(t)} = \Phi\left(W^T \vec{h}^{(t-1)}\right) = \Phi\left(W^{\mathrm{T}} \Phi\left(W^T \vec{h}^{(t-1)}\right)\right) = \cdots$$

$$= \Phi\left(W^T \Phi\left(W^{\mathrm{T}} \Phi\left(W^T \Phi\left(\ldots\left(\Phi\left(W^T \vec{h}^{0}\right)\right)\right)\right)\right)\right)$$

  to simplify things let's assume this is ReLU and we only analyze the >0 part of the composition. Then this expression becomes $\vec{h}^{(t)} = (W^T)^t \vec{h}^{(0)}$.

- If W is diagonalizable then $W = Q\Lambda Q^T$ for some Q and

$$(W^T)^t = Q^T \Lambda^{\mathrm{t}} Q$$

Where do Gradients Problems Originate (cont.)

- So $\vec{h}^{(t)} = (W^T)^t \vec{h}^{(0)}$ and $(W^T)^t = Q^T \Lambda^t Q$
- Let's assume that W is square and Q is orthonormal basis. Then $\left[\vec{h}^{(0)}\right]_Q = Q^T \vec{h}^{(0)}$ is the hidden 0 layer in new coordinates.
- Suppose now there are some eigenvalues in Λ that are >> 1 and others are <1. Then components of $\left[\vec{h}^{(0)}\right]_Q$ that correspond to eigenvalues bigger than 1 will explode and the ones that are smaller than 1 will vanish
- If the difference between largest eigenvalue and others is big then any component of $\vec{h}^{(0)}$ that is not aligned with the largest eigenvector will eventually be (relatively) discarded.
- Since the backpropagation in this case is a multiplication by transpose matrix (Jacobian is transpose in this case) it is dominated by largest eigenvalue as well!

Where do Gradients Problems Originate (cont.)

- Denote $J = J_{\vec{h}}^{(t)} = \frac{\partial \vec{h}^{(t)}}{\partial \vec{h}^{(t-1)}}$ - Jacobian of backpropagation and assume it is the same at each step. Let now its spectral radius be $r = \max_{\lambda_i} |\lambda|$.
- On backpropagation start with true gradient vector $\vec{g}$ then after n backpropagations we have $J^n \vec{g}$.
- We start with perturbed gradient vector $(\vec{g} + \delta\vec{v})$ then we end up with $J^n g + J^n \delta\vec{v}$ and if $\vec{v}$ is a unit eigenvector of $\operatorname{argmax}_{\lambda_i} |\lambda|$, then $J^n \delta\vec{v} = r^n \delta$.
- So if $r \gg 1$ then even starting with small perturbation it explodes exponentially
- The gradient g may not have exploded that much it wasn't colinear to v even. So we may end up with backpropagation of $\vec{g}$ being comparable to $r^n \delta$ even though $\|\delta\vec{v}\|$ was small

Where do Gradients Problems Originate (cont.)

- For $J = W^T$ if $r \ll 1$ then W is *contractive*, if $r > 1$then it is *expanding*. If J is contractive in the limit gradient vanishes, if it is expanding then in the limit it explodes
- The Jacobian matrix tells us how a small change of $h^{(t)}$ propagates one step forward, or equivalently, how the gradient on $h^{(t+1)}$ propagates one step backward
- With a nonlinear activation Jacobian changes at each step. But the type of dynamics remains.

Activation Propensity to Vanishing Gradients

- Partial derivative of sigmoid with output $o \Rightarrow o(1 - o)$.
  - Maximum value at o = 0.5 of 0.25.
  - For 10 layers, the activation function alone will multiply by less than $(0.25)^{10} \approx 10^{-6}$.
- At extremes of output values, the partial derivative is close to 0, which is called *saturation*.
- The tanh activation function with partial derivative $(1 - o^2)$ has a maximum value of 1 at $o = 0$, but saturation will still cause problems.

Lecture Overview

1. BPTT
2. Complex Networks
3. Exploding and Vanishing Gradients
4. Fixing Long Term Dependencies

Echo State Network

- The strategy is simply to fix hidden-to hidden weight at relatively small spectral radius – usually less than 3
- If set so information is carried forward through time but does not explode due to the stabilizing effect of saturating nonlinearities like tanh.
- The backpropagation learns other weights but not hidden-to-hidden
- It is still possible for back-propagation to lead to explode even when forward propagation has bounded dynamics

Adding Skip Connections

- The strategy: add direct connections from variables in the distant past to variables in the present
  - In an 'normal' RNN a recurrent connection goes from a unit at time t to a unit at time $t + 1$.
  - Instead connect from t to $t + d$ AND to $t + 1$. Thus some backpropagation paths have length at most $\frac{\tau}{d}$
- Since gradients may vanish or explode exponentially with respect to the number of time steps. Shorter paths mitigate the problem
- In RNN with skip connections one has both 1 step and d-skip connections, so gradient can still explode but ti is less likely to vanish which is a more often problem of the two

Leaky Units

This method beats vanishing gradients as well since it adds smooth shorter paths contribution

- The leaky unit is defined using an additional meta-parameter $\alpha \in (0, 1)$ and with linear connections to themselves, for example for a given activation function Φ leaky unit can be defined as

$$\Phi^{Modified}\left(x^{(t)}, h^{(t-1)}\right) = \frac{\alpha}{k}\sum_{i=1}^{k} \Phi\left(x^{(t-i)}, h^{(t-i-1)}\right) + (1-\alpha)\Phi\left(x^{(t)}, h^{(t-1)}\right)$$

- When α is near 1, the output remembers information about the past, when it is close to 0 it is close to current output with minor adjustment for the past
- How to set the time constants (in this case k) used by leaky units?
  - Option 1: is to choose at initialization time and keep constant
  - Option2: learn them same as other parameters

Removing Connections

- The strategy: *remove* length-one connections and replace them with longer connections. Units modified in such a way are forced to operate on a longer time scale.
- Ways in which groups of units can be forced to operate at different time scales:
  1. Combine leaky units with non-leaky units operating on 1 scale and units operating on a some d-scale
  2. have explicit, discrete updates taking place at different times, with a different frequency for different groups of units

Reading

- Ch. 3.4.1-3.4.3
- Ch. 7.2.2-7.2.4.
- Ch 7.3. (intro only – before 7.3.1)
- Ch. 7.4